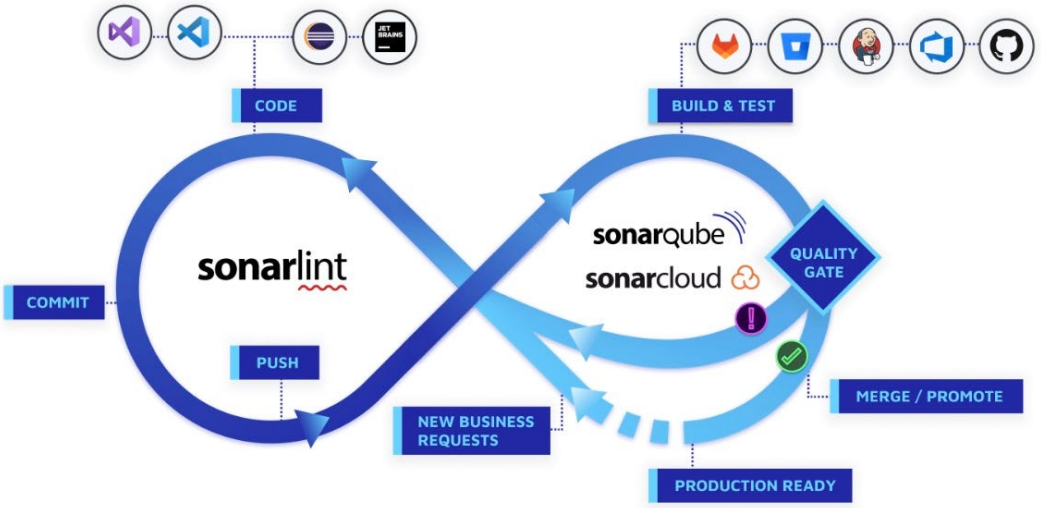
Static Code Analysis with SonarQube

What is `SonarQube`?

- **SonarQube** is a widely-used tool for **continuous inspection** of code quality, aimed at detecting **bugs, vulnerabilities, and code smells**. Supporting numerous programming languages such as **Java, Python, JavaScript, PHP**, and more, it integrates easily into development workflows.
- `SonarQube` helps teams maintain **high standards of quality and security** by offering detailed dashboards to track issues in real-time, alongside key metrics like **code coverage**.

▼ How `SonarQube` Works



`SonarQube` integrates with your build tools and CI/CD pipelines for automated analysis. The workflow typically follows these steps:

1. **Code Push**: Developers commit code to a repository (GitHub, GitLab, etc.).
2. **Code Scan**: SonarQube scans the codebase, applying rules to identify bugs, smells, and security vulnerabilities.
3. **Dashboard Reports**: Issues are displayed in the SonarQube web interface, organized by severity and type.
4. **Quality Gates**: Automatically determine if code meets defined thresholds for security, code quality, and technical debt.

▼ Key Features

1. **Bug & Vulnerability Detection**: Identifies logical errors and security risks (e.g., SQL injection).
2. **Code Duplication**: Highlights repeated code, increasing technical debt.
3. **Code Smells**: Pinpoints potential maintainability issues (e.g., overly complex methods).
4. **Test Coverage**: Measures how much of the code is covered by unit tests, ensuring critical parts are tested.

▼ Benefits of `SonarQube`

- **Early Detection of Issues**:

> `SonarQube` catches bugs and vulnerabilities in the early stages of development, preventing critical issues from making their way into production.

- **CI/CD Pipeline Integration**:

> `SonarQube` can be integrated into any CI/CD pipelines. It helps enforce quality standards on every code push.

- **Code Refactoring Guidance**:

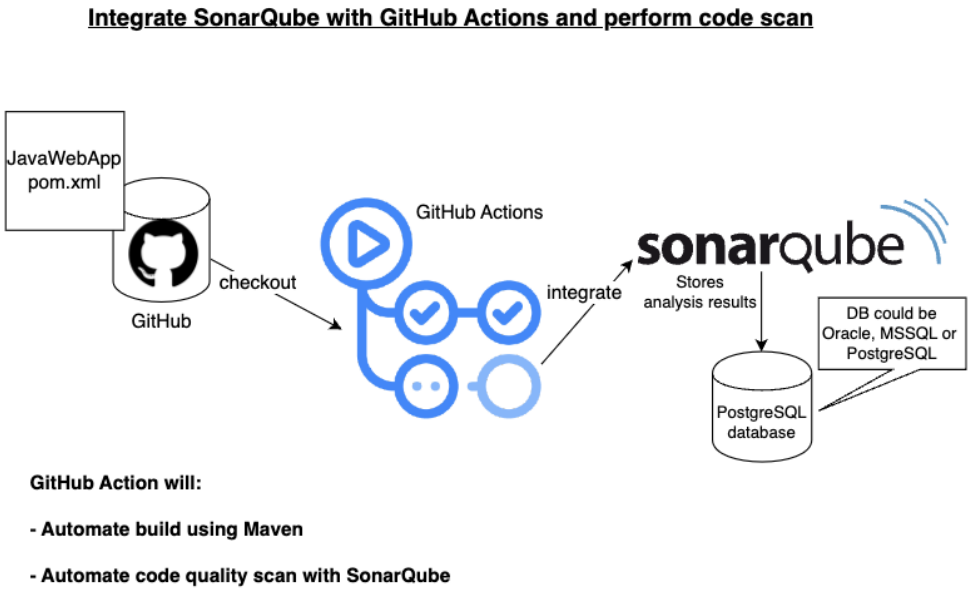
> `SonarQube` suggests improvements to the code and points out areas that may require refactoring, helping developers improve code quality continuously.

- **Security**:

> `SonarQube` identifies potential security vulnerabilities and compliance issues - (e.g., it can detect vulnerable code patterns, such as the use of weak encryption, improper input validation, and SQL injection risks).

▼ How to Use `SonarQube`?

A. Installation



For **Windows/MacOS/Linux**, the setup steps are similar:

1. Download `SonarQube` (Community Edition)
2. Extract the package.
3. Start SonarQube using the appropriate command for your OS:
   - for Windows: `bin/windows-x86-64/StartSonar.bat`
   - for macOS: `bin/macosx-universal-64/sonar.sh start`
4. Navigate to `http://localhost:9000` to access the dashboard.

B. Configuring `SonarQube` for Spring Boot

- We can integrate SonarQube with a **Spring Boot** project by using the **SonarQube Maven plugin**.
- Add the plugin to your `pom.xml` file, run the Maven build, and SonarQube will automatically analyze your code:

```
<build>
   <plugins>
       <plugin>
           <groupId>org.sonarsource.scanner.maven</
groupId>
           <artifactId>sonar-maven-plugin</artifact
Id>
```

```
            <version>3.9.1.2184</version>
        </plugin>
    </plugins>
</build>
```

- Run the following command to analyse the project:

```
mvn sonar:sonar
```

▼ `SonarQube` Best Practices

- **Integrate Early**: Ensure that static code analysis is part of the development pipeline from the very start.
- **Automate**: Use CI/CD tools like **GitHub Actions** or **Jenkins** to run SonarQube scans automatically on each code commit or pull request.
- **Review Regularly**: Regularly review SonarQube reports and address any critical vulnerabilities, code smells, or duplication issues.
- **Customise Rules**: Tailor the SonarQube rules to meet the specific needs of your project.

▼ Practical Tips for Using `SonarQube`

- **Run Regularly**: Add static code analysis to your **CI pipeline** for automatic checks with every code push. This prevents technical debt from accumulating over time.
- **Focus on High Severity Issues**: Prioritise critical bugs and security vulnerabilities flagged by `SonarQube`, which may affect the stability or security of your application.
- **Use Quality Gates**: Quality gates ensure the code meets the defined thresholds for bugs, vulnerabilities, and other metrics before it is merged or deployed.
- **Track Technical Debt**: `SonarQube` provides a "technical debt" metric, which estimates the effort required to fix issues in the code.

▼ Preparing `SonarQube` for `hackApp`

Step 1: Configure the HackApp for SonarQube

Before you start the static analysis, ensure your **pom.xml** is configured to use `SonarQube`. Here's what you need to do:

- In your `hackApp` project, open the **pom.xml** file.
- Add the `SonarQube` Maven plugin to enable code analysis:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.sonarsource.scanner.maven</group
Id>
            <artifactId>sonar-maven-plugin</artifactId>
            <version>3.9.1.2184</version>
        </plugin>
    </plugins>
</build>
```

- Also, include the following properties to connect `SonarQube` to your local instance (or a remote server if applicable):

```
<properties>
    <sonar.projectKey>your_project_key</sonar.projectKey>
    <sonar.host.url>http://localhost:9000</sonar.host.url
>
    <sonar.login>your_sonarqube_token</sonar.login>
</properties>
```

- The `sonar.login` property is your `SonarQube` **token** (which you can generate in the `SonarQube` dashboard). The `projectKey` uniquely identifies your project on the `SonarQube` dashboard.

▼ Running SonarQube Locally for `hackApp`

Step 2: Start the `SonarQube` Server

- Ensure that `SonarQube` is running on your machine. If not, follow these steps:
    1. Download the `SonarQube` package from here.
    2. Extract the downloaded package.

3. Start SonarQube:
    - On **Windows**: Run `StartSonar.bat` from the `bin/windows-x86-64` folder.
    - On **Mac/Linux**: Run `sonar.sh start` from the `bin/macosx-universal-64` or the equivalent directory for Linux.
- Verify SonarQube is running by navigating to `http://localhost:9000` in your browser.

▼ Running the Static Code Analysis on `hackApp`

Step 3: Running the Analysis

- In the **HackApp** root directory, open a terminal and execute the following Maven command:

```
mvn clean verify sonar:sonar
```

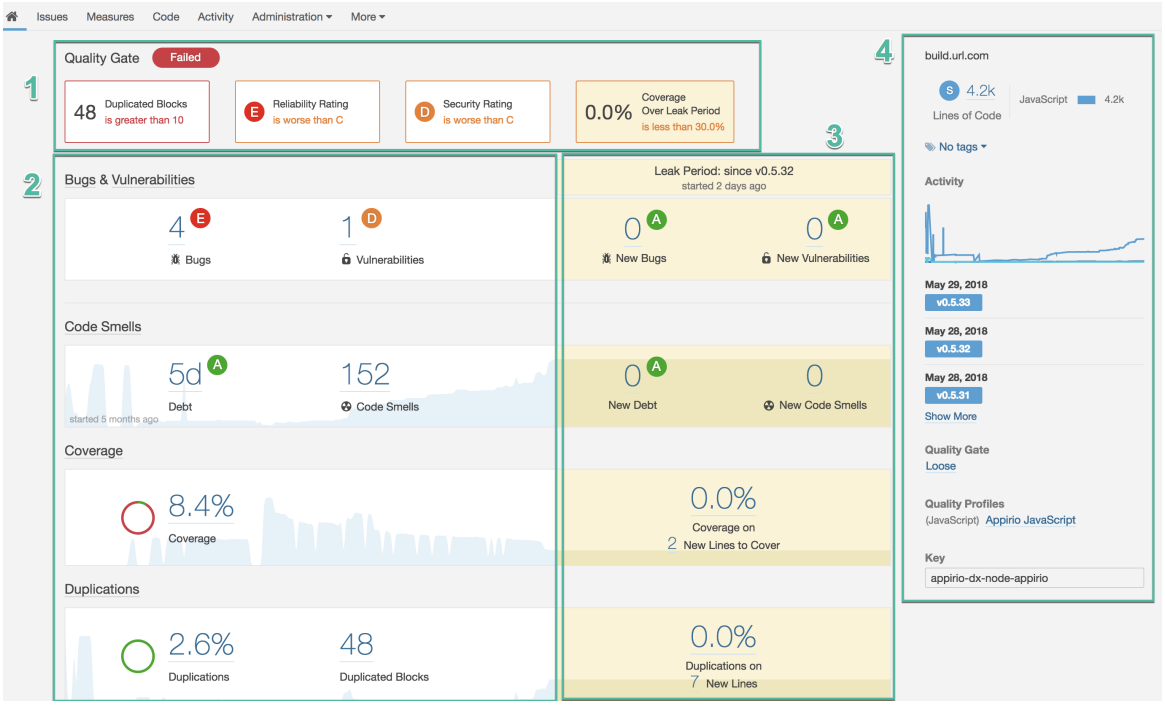
- This will:
    1. Clean and build your project (`clean verify`).
    2. Run the SonarQube static analysis (`sonar:sonar`).
- The Maven command tells `SonarQube` to scan the project's source code and push the results to your running `SonarQube` instance. The plugin connects to the `SonarQube` server (at the URL specified in **pom.xml**) and uploads the analysis results.

▼ Viewing SonarQube Results for HackApp

Step 4: Checking the Analysis Results

- Once the analysis is complete, go to your `SonarQube` dashboard at `http://localhost:9000`.
- Find the project you just analysed (based on the `projectKey`).
- Review key metrics:
    - **Bugs**: Issues that can cause the code to malfunction.
    - **Code Smells**: Maintainability issues, such as poor coding practices or duplications.

- **Vulnerabilities**: Security risks like potential SQL injection or XSS risks.



The dashboard will show you the analysis results of **HackApp**. You can see detailed reports, recommendations for fixes, and metrics like **code duplication**, **code coverage**, and **technical debt**. You can use this information to improve your code quality iteratively.

▼ Automating `SonarQube` with GitHub Actions

▼ ⚠ Cautions

- If you're using **SonarCloud**, configure the **SonarCloud URL** (`https://sonarcloud.io`) and use the **SonarCloud token**.
- If you have a self-hosted SonarQube instance, ensure it's **publicly accessible** and update the `sonar.host.url` in both your `pom.xml` and GitHub Actions workflow.
- Running SonarQube locally (`localhost:9000`) within GitHub Actions is **not possible** unless you run SonarQube as a service in Docker inside the runner.

▼ Step 5: GitHub Actions Workflow for `SonarQube`

- Automate this process by integrating `SonarQube` into the GitHub Actions pipeline, ensuring that every time code is pushed to the `hackApp` repository, the static code analysis runs automatically.
- Here's an example GitHub Actions workflow (`.github/workflows/sonarqube.yml`):

```
name: SonarQube Analysis

on:
  push:
    branches:
      - main

jobs:
  sonarQube:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          java-version: '17'

      - name: Build with Maven
        run: mvn clean verify

      - name: Run SonarQube analysis
        run: mvn sonar:sonar
        env:
          SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL
}}
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

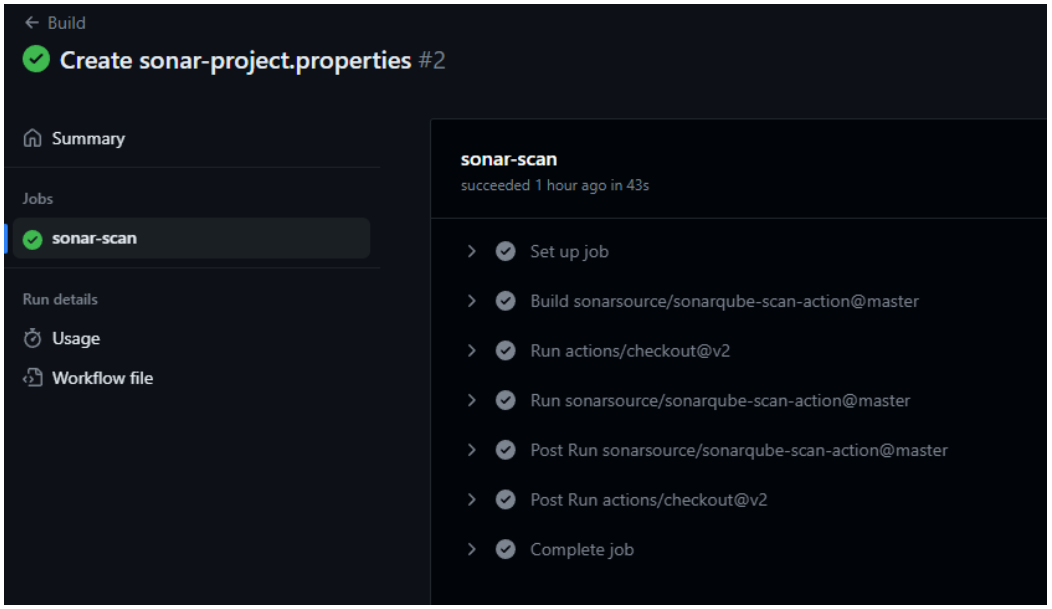
▼ Step 6: Adding `SonarQube` Secrets in GitHub

- **Step 1**: Go to your GitHub repository settings.
- **Step 2**: Navigate to **Settings** > **Secrets and variables > Actions**.
- **Step 3**: Add two secrets:
  - **SONAR_HOST_URL**: Set this to `http://localhost:9000` (or the URL where SonarQube is hosted).
  - **SONAR_TOKEN**: Generate a token in SonarQube under **My Account > Security** and paste it here.

This workflow ensures that the code is analyzed every time a developer pushes changes to the **main** branch. It connects to the SonarQube instance using the provided token and URL (stored as GitHub **Secrets** for security).

▼ Step 7: Running Static Code Analysis

- Push code changes to GitHub.
- GitHub Actions will automatically trigger the SonarQube analysis workflow.
- Check the **Actions** tab in your GitHub repository to view the progress.
- After the analysis is complete, navigate to the SonarQube dashboard at `http://localhost:9000` to view the results.



This consolidated workflow should be appropriate for running **static code analysis** on `hackApp` using `SonarQube`, both locally and with CI/CD through GitHub Actions.

▼ Introduction to `SonarCloud`

What is SonarCloud?

- A cloud-based version of `SonarQube` that provides code analysis directly from GitHub, Bitbucket, GitLab, or Azure repositories.
- Scans code for bugs, security vulnerabilities, and code smells in over 20 languages.
- No need for local installation—easily integrates with CI/CD pipelines.

Why Use SonarCloud?

- Simplifies setup by eliminating the need for local infrastructure.
- Free for open-source projects and provides commercial tiers for private repositories.
- Ideal for real-time code quality checks in cloud-based development workflows.

Key Features:

- Real-time integration with GitHub Actions for automatic quality checks.
- Visualised dashboards for code quality metrics.
- Supports static analysis for various languages and technologies (e.g., Java, JavaScript, Python, and more).

▼ `SonarCloud` Setup with GitHub Actions

How to Set Up `SonarCloud` in GitHub Actions:

1. **Create a `SonarCloud` Account**:
   - Visit SonarCloud.io and sign up (GitHub login is recommended).
2. **Create a Project on SonarCloud**:
   - Select your repository from GitHub when prompted during the setup.
3. **Generate a Token**:

- Navigate to *Account Settings* in SonarCloud and generate a token to allow GitHub to interact with SonarCloud.

4. **Add Token to GitHub**:
   - In your GitHub repository, navigate to *Settings* → *Secrets* → *Actions* → *New repository secret*.
   - Name it `SONAR_TOKEN` and paste the generated token.
5. **Update GitHub Actions Workflow**:
   - Modify your existing GitHub Actions `.yml` file to include the SonarCloud analysis. Example:

```
name: SonarCloud
on:
  push:
    branches:
      - master
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
  build:
    name: Build and analyze
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0  # Shallow clones should be dis
abled for a better relevancy of analysis
      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          java-version: 17
          distribution: 'zulu' # Alternative distributio
n options are available.
      - name: Cache SonarCloud packages
        uses: actions/cache@v4
        with:
          path: ~/.sonar/cache
```

```
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      - name: Cache Maven packages
        uses: actions/cache@v4
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/po
m.xml') }}
          restore-keys: ${{ runner.os }}-m2
      - name: Build and analyze
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # N
eeded to get PR information, if any
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: mvn -B verify org.sonarsource.scanner.mave
n:sonar-maven-plugin:sonar -Dsonar.projectKey=eiramlan_h
ackapp-sonarqube
```

Review SonarCloud Results:

- After pushing changes or opening a pull request, GitHub Actions will trigger the scan, and the results will appear on the SonarCloud dashboard.